

Deep Learning on Lie Groups for Skeleton-based Action Recognition

Zhiwu Huang, Chengde Wan, Thomas Probst, Luc Van Gool
Computer Vision Laboratory, D-ITET, ETH Zurich

{zhiwu.huang, wanc, probstt, vangool}@vision.ee.ethz.ch

Abstract

In recent years, skeleton-based action recognition has become a popular 3D classification problem. State-of-the-art methods typically first represent each motion sequence as a high-dimensional trajectory on a Lie group with an additional dynamic time warping, and then shallowly learn favorable Lie group features. In this paper we incorporate the Lie group structure into a deep network architecture to learn more appropriate Lie group features for 3D action recognition. Within the network structure, we design rotation mapping layers to transform the input Lie group features into desirable ones, which are aligned better in the temporal domain. To reduce the high feature dimensionality, the architecture is equipped with rotation pooling layers for the elements on the Lie group. Furthermore, we propose a logarithm mapping layer to map the resulting manifold data into a tangent space that facilitates the application of regular output layers for the final classification. Evaluations of the proposed network for standard 3D human action recognition datasets clearly demonstrate its superiority over existing shallow Lie group feature learning methods as well as most conventional deep learning methods.

1. Introduction

Due to the development of depth sensors, 3D human activity analysis [26, 44, 22, 42, 40, 3, 41, 36, 43, 25, 34, 17] has attracted more interest than ever before. Recent manifold-based approaches are quite successful at 3D human action recognition thanks to their view-invariant manifold-based representations for skeletal data. Typical examples include shape silhouettes in Kendall’s shape space [39, 3], linear dynamical systems on the Grassmann manifold [38], histograms of oriented optical flow on a hypersphere [11], and pairwise transformations of skeletal joints on a Lie group [40, 3, 41]. In this paper, we focus on studying manifold-based approaches [40, 3, 41] to learn more appropriate Lie group representations of skeletal action data, that have achieved state-of-the-art performances for some 3D human action recognition benchmarks.

As studied in [40, 3, 41], Lie group feature learning methods often suffer from speed variations (i.e., temporal misalignment), that tend to deteriorate classification accuracy. To handle this issue, they typically employ dynamic time warping (DTW), as originally used in speech processing [29]. Unfortunately, such process costs additional time, and also results in a two-step system that typically performs worse than an end-to-end system. Moreover, such Lie group representations for action recognition tend to be extremely high-dimensional, in part because the features are extracted per skeletal segment and then stacked. As a result, any computation on such non-linear trajectories is expensive and complicated. To address this problem, [40, 3, 41] attempt to first flatten the underlying manifold via tangent approximation or rolling maps, and then exploit SVM or PCA-like method to learn features in the resulting flattened space. Although these methods achieve some success, they merely adopt shallow linear learning schemes, yielding sub-optimal solutions on the specific non-linear manifolds.

Deep neural networks have shown their great power in learning compact and discriminative representations for images and videos, thanks to their ability to perform non-linear computations and the effectiveness of gradient-descent training with backpropagation. This has motivated us to build a deep neural network architecture for representation learning on Lie groups. In particular, inspired by the classical manifold learning theory [37, 35, 4, 12, 20, 19], we equip the new network structure with rotation mapping layers, with which the input Lie group features are transformed to new ones with better alignment. As a result, the effect of speed variations can be appropriately mitigated. In order to reduce the high dimensionality of the Lie group features, we design special pooling layers to compose them in terms of bone-based and skeleton-based ones, respectively. As the output data reside on non-linear manifolds, we also propose a Riemannian computation layer, whose outputs could be fed into any regular output layers such as a softmax layer. In short, our main contributions are:

- A novel neural network architecture is introduced to deeply learn more desirable Lie group representations for the problem of skeleton-based action recognition.

- The proposed network provides a paradigm to incorporate the Lie group structure into deep learning, which generalizes the traditional neural network paradigm to non-Euclidean Lie groups.
- To train the network within the backpropagation framework, a variant of stochastic gradient descent optimization is exploited in the context of Lie groups.

2. Relevant Work

Already quite some works [45, 33, 2, 28, 32, 14, 15] have applied aspects of Lie group theory to deep neural networks. For example, [32] investigated how stability properties of a continuous recursive neural network can be altered within neighbourhoods of equilibrium points by the use of Lie group projections operating on the synaptic weight matrix. [14] studied the behavior of unsupervised neural networks with orthonormality constraints, by exploiting the differential geometry of Lie groups. In particular, two sub-classes of the general Lie group learning theories were studied in detail, tackling first-order (gradient-based) and second-order (non-gradient-based) learning. [15] introduced deep symmetry networks (symnets), a generalization of convolutional networks that forms feature maps over arbitrary symmetry groups that are basically Lie groups. The symnets utilize kernel-based interpolation to tractably tie parameters and pool over symmetry spaces of any dimension.

Moreover, recently some deep learning models have emerged [10, 7, 27, 24, 18] that deal with data in a non-Euclidean domain. For instance, [10] proposed a spectral version of convolutional networks to handle graphs. It exploits the notion of non shift-invariant convolution, relying on the analogy between the classical Fourier transform and the Laplace-Beltrami eigenbasis. [24] developed a scalable method for treating an arbitrary spatio-temporal graph as a rich recurrent neural network mixture, which can be used to transform any spatio-temporal graph by employing a certain set of well-defined steps. For shape analysis, [27] proposed a ‘geodesic convolution’ on local geodesic coordinate systems to extract local patches on the shape manifold. This approach performs convolutions by sliding a window over the manifold, and local geodesic coordinates are used instead of image patches. To deeply learn symmetric positive definite (SPD) matrices - used in many tasks - [18] developed a Riemannian network on the manifold of SPD matrices, with some layers specially designed to deal with such structured matrices.

In summary, such works have applied some theories of Lie groups to regular networks, and even generalized the common networks to non-Euclidean domains. Nevertheless, to the best of our knowledge, this is the first work that studies a deep learning architecture on Lie groups to handle the problem of skeleton-based action recognition.

3. Lie Group Representation for Skeletal Data

Let $S = (V, E)$ be a body skeleton, where $V = \{v_1, \dots, v_N\}$ denotes the set of body joints, and $E = \{e_1, \dots, e_M\}$ indicates the set of edges, i.e. oriented rigid body bones. As studied in [40, 3, 41], the relative geometry of a pair of body parts e_n and e_m can be represented in a local coordinate system attached to the other. The local coordinate system of body part e_n is calculated by rotating with minimum rotation so that its starting joint becomes the origin and it coincides with the x -axis. Then we can compute the rotation matrix $R_{m,n}$ ($R_{m,n}^T R_{m,n} = R_{m,n} R_{m,n}^T = I_n, |R_{m,n}| = 1$) from e_m to the local coordinate system of e_n . Specifically, we can firstly calculate the axis-angle representation (ω, θ) for the rotation matrix $R_{m,n}$ by

$$\omega = \frac{\hat{e}_m * \hat{e}_n}{\|\hat{e}_m * \hat{e}_n\|}, \quad (1)$$

$$\theta = \arccos(\hat{e}_m \bullet \hat{e}_n). \quad (2)$$

where $*$, \bullet are outer and inner products respectively, and \hat{e}_m, \hat{e}_n indicate the normalized 3D vectors of the two edges e_m, e_n respectively. Subsequently, we can transform the axis-angle representation to the rotation matrix $R_{m,n}$. In the same way, the rotation matrix $R_{n,m}$ from e_n to the local coordinate system of e_m can be computed. As a result, both $R_{m,n}$ and $R_{n,m}$ are used to represent the relative geometry between e_m and e_n . Accordingly, a skeleton S at time instance t is represented by $R_{1,2}(t) \times R_{2,1}(t) \times \dots \times R_{M-1,M}(t) \times R_{M,M-1}(t)$, where M is the number of body parts. Note that there are $2 \times C_M^2$ (C is the combination computation) rotation matrices in total for a skeleton.

The set of $n \times n$ rotation matrices in \mathbb{R}^n forms the special orthogonal group SO_n which is actually a matrix Lie group [21, 9, 16]. Accordingly, each motion sequence of a moving skeleton is represented with a curve on the Lie group $SO_3 \times \dots \times SO_3$. It is known that the matrix Lie group is endowed with a Riemannian manifold structure that is differentiable. Hence, at each point R_0 on SO_n , one can derive the tangent space $T_{R_0}SO_n$ that is a vector space spanned by the set of skew-symmetric matrices. When the anchor point is the identity matrix $I_n \in SO_n$, the resulting tangent space is known as the Lie algebra so_n . As the tangent spaces are equipped with the inner product, the Riemannian metric on SO_n can be defined by the Frobenius inner product:

$$\langle A_1, A_2 \rangle = \text{trace}(A_1^T A_2), A_1, A_2 \in T_{R_0}SO_n. \quad (3)$$

The logarithm map \log_{R_0} and exponential map \exp_{R_0} at R_0 on SO_n associated with the Riemannian metric can be expressed in terms of the usual matrix logarithm \log and exponential \exp as

$$\log_{R_0}(R_1) = \log(R_1 R_0^T) \text{ with } R_0, R_1 \in SO_n, \quad (4)$$

$$\exp_{R_0}(A_1) = \exp^{A_1 R_0^T} \text{ with } A_1 \in T_{R_0}SO_n. \quad (5)$$

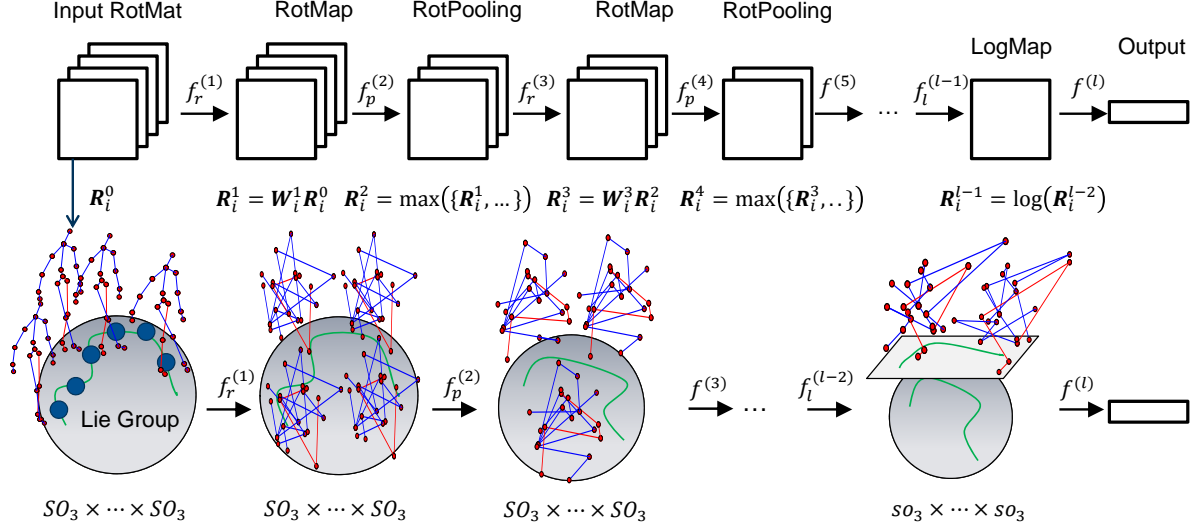


Figure 1. Conceptual illustration of the proposed Lie group Network (LieNet) architecture. The data space of each RotMap/RotPooling layer in the deep network architecture corresponds to one Lie group, while the weight spaces of the RotMap layers are Lie groups as well.

4. Lie Group Network for Skeleton-based Action Recognition

For the problem of skeleton-based action recognition, we build a deep network architecture to learn the Lie group representations for skeletal data. The new network architecture is dubbed as LieNet, where each input is an element on the Lie Group. Like other convolutional networks (ConvNets), the LieNet also exhibits fully connected convolution-like layers and pooling layers, named rotation mapping (RotMap) layers and rotation pooling (RotPooling) layers respectively. In particular, the proposed RotMap layers perform transformations on input rotation matrices to generate new rotation matrices, which have the same manifold property, and are expected to be aligned more accurately for more reliable matching. The RotPooling layers aim to pool the resulting rotation matrices at a bone-based and a skeleton-based level such that the Lie group feature dimensionality can be reduced. Since the rotation matrices reside on non-Euclidean manifolds, we have to design a layer named logarithm mapping (LogMap) layer, to perform the Riemannian computations on. This transforms the rotation matrices into the usual skew-symmetric matrices, which lie in Euclidean space and hence can be fed into any regular output layers. The structure of the proposed LieNet is shown in Fig.1.

4.1. RotMap Layer

As well-known from classical manifold learning theory [37, 35, 4, 12, 20, 19], one can learn or preserve the original data structure to faithfully maintain geodesic distances for better classification. Accordingly, we design a RotMap

layer to transform the input rotation matrices to new ones that are more suitable for the final classification. Formally, the RotMap layers adopt a rotation mapping f_r as

$$\begin{aligned} f_r^{(k)}(R_1^{k-1} \times R_2^{k-1} \dots \times R_M^{k-1}; W_1^k, W_2^k, \dots, W_M^k) \\ = (W_1^k R_1^{k-1}) \times (W_2^k R_2^{k-1}) \dots \times (W_M^k R_M^{k-1}) \\ = R_1^k \times R_2^k \dots \times R_M^k \end{aligned} \quad (6)$$

where $\hat{M} = 2 \times C_M^2$ (M is the number of body bones in one skeleton, C is the combination computation), $R_1^{k-1} \times R_2^{k-1} \dots \times R_M^{k-1} \in SO_3 \times SO_3 \dots \times SO_3$ is the input Lie group feature (i.e., product of rotation matrices) for one skeleton in the k -th layer, $W_i^k \in \mathbb{R}^{3 \times 3}$ is the transformation matrix (connection weights), and $R_1^k \times R_2^k \dots \times R_M^k$ is the resulting Lie group representation. Note that although there is only one transformation matrix for each rotation matrix, it would be easily extended with multiple projections for each input. To ensure the matrix $R_1^k \times R_2^k \dots \times R_M^k$ becomes a valid product of rotation matrices residing on $SO_3 \times SO_3 \dots \times SO_3$, the transformation matrices $W_1^k, W_2^k, \dots, W_M^k$ are all basically required to be rotation matrices. In other words, both the data space and the weight space on each RotMap layer correspond to one product manifold $SO_3 \times SO_3 \dots \times SO_3$.

Since the RotMap layers are designed to work together with the classification layer, each resulting skeleton representation is tuned for more accurate classification in an end-to-end deep learning manner. In other words, the major purpose of designing the RotMap layers is to align the Lie group representations of a moving skeleton for more faithful matching.

4.2. RotPooling Layer

In order to reduce the complexity of deep models, it is typically useful to reduce the size of the representations to decrease the amount of parameters and computation in the network. For this purpose, it is common to insert a pooling layer in-between successive convolutional layers in a typical ConvNet architecture. The pooling layers are often designed to compute statistics in local neighborhoods, such as the average energy or maximum activation.

The same type of layers can be defined in the setting of the proposed network by providing the equivalent notion of neighborhood. Since the input and output of this pooling layer are both rotation matrices, we call this kind of layers as rotation pooling (RotPooling) layer. For the RotPooling, we propose two different concepts of neighborhood in this work. The first one is on the bone-based level. As shown in Fig.2(a)→(b), we first pool the Lie group features on each pair of basic bones e_m, e_n in the i -th frame, which is represented by the two rotation matrices $\mathbf{R}_{m,n}^{k-1,i}, \mathbf{R}_{n,m}^{k-1,i}$ (here $k-1$ is the order of the layer) as aforementioned. Then, as depicted in Fig.2(b)→(c), we can perform pooling on the adjacent bones that belong to the same group (here, we can define five part groups, i.e., torso, two arms and two legs, of the body). However, the second step would inevitably result in a serious spatial misalignment problem, and thus lead to bad matching performances. Therefore, we finally only adopt the first step pooling. In this setting, the function of the max pooling is given by

$$\begin{aligned} f_p^{(k)}(\mathbf{R}_{m,n}^{k-1,i}, \mathbf{R}_{n,m}^{k-1,i}) &= \max(\mathbf{R}_{m,n}^{k-1,i}, \mathbf{R}_{n,m}^{k-1,i}) \\ &= \begin{cases} \mathbf{R}_{m,n}^{k-1,i}, & \text{if } \Theta(\mathbf{R}_{m,n}^{k-1,i}) > \Theta(\mathbf{R}_{n,m}^{k-1,i}), \\ \mathbf{R}_{n,m}^{k-1,i}, & \text{otherwise,} \end{cases} \end{aligned} \quad (7)$$

where $\Theta(\cdot)$ is the representation of the given rotation matrix such as quaternion, Euler angle or Euler axis-angle. For example, the Euler axis ω and angle θ representations are typically calculated by

$$\theta(\mathbf{R}_{n,m}) = \arccos\left(\frac{\text{trace}(\mathbf{R}_{n,m}) - 1}{2}\right), \quad (8)$$

$$\omega(\mathbf{R}_{n,m}) = \frac{1}{2\sin(\theta(\mathbf{R}_{n,m}))} \begin{pmatrix} \mathbf{R}_{n,m}(3,2) - \mathbf{R}_{n,m}(2,3) \\ \mathbf{R}_{n,m}(1,3) - \mathbf{R}_{n,m}(3,1) \\ \mathbf{R}_{n,m}(2,1) - \mathbf{R}_{n,m}(1,2) \end{pmatrix}, \quad (9)$$

where $\mathbf{R}_{n,m}(i,j)$ is the i -th row, j -th column element of $\mathbf{R}_{n,m}$. Unfortunately, except the angle representation, it is non-trivial to define an ordering relation for a quaternion or an axis-angle representation. Accordingly, in this paper, we finally adopt the angle form Eqn.8 of rotation matrices and its simple ordering relation to calculate the function $\Theta(\cdot)$.

The other pooling scheme is on the skeleton-based level. As shown in Fig.2 (c)→(d), the aim of the skeleton-based

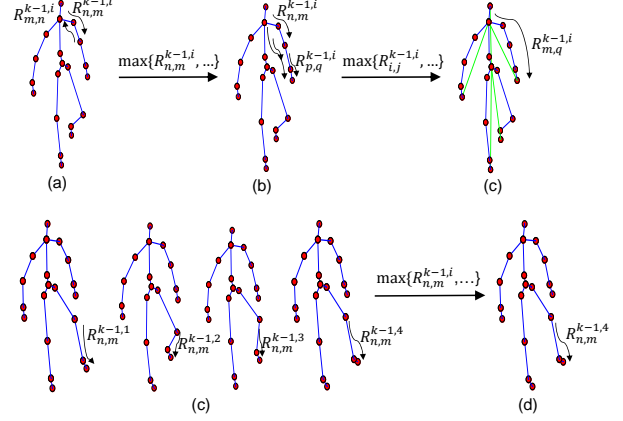


Figure 2. Conceptual illustration of bone-based pooling scheme (a)→(b)→(c) and skeleton-based pooling scheme (c)→(d).

pooling is to obtain more compact representations for a motion sequence. This is because a sequence often contains many frames and thus gives rise to the problem of extremely high-dimensional representations. Thus, pooling in the temporal domain could reduce the model complexity. Formally, the function of this kind of max pooling is defined as

$$\begin{aligned} f_p^{(k)}(\{\mathbf{R}_{1,2}^{k-1,1} \times \dots \times \mathbf{R}_{M-1,M}^{k-1,1}, \dots, \mathbf{R}_{1,2}^{k-1,p} \times \dots \times \mathbf{R}_{M-1,M}^{k-1,p}\}) \\ = \max(\{\mathbf{R}_{1,2}^{k-1,1}, \dots, \mathbf{R}_{1,2}^{k-1,p}\}) \times \dots \\ \times \max(\{\mathbf{R}_{M-1,M}^{k-1,1}, \dots, \mathbf{R}_{M-1,M}^{k-1,p}\}), \end{aligned} \quad (10)$$

where M is the number of body parts in one skeleton, p is the number of skeleton frames for pooling, and the function $\max(\cdot)$ is defined in the way of Eqn.7.

4.3. LogMap Layer

Classification of curves on the Lie group $SO_3 \times \dots \times SO_3$ is a complicated task due to the non-Euclidean nature of the underlying space. To address the problem as in [41], we design the logarithm map (LogMap) layer to flatten the Lie group $SO_3 \times \dots \times SO_3$ to its Lie algebra $so_3 \times \dots \times so_3$. Accordingly, by using the logarithm map Eqn.4, the function of this layer can be defined as

$$\begin{aligned} f_l^{(k)}(\mathbf{R}_1^{k-1} \times \mathbf{R}_2^{k-1} \dots \times \mathbf{R}_M^{k-1}) \\ = \log(\mathbf{R}_1^{k-1}) \times \log(\mathbf{R}_2^{k-1}) \dots \times \log(\mathbf{R}_M^{k-1}). \end{aligned} \quad (11)$$

One typical approach to calculate the logarithm map is to use the approach $\log(\mathbf{R}) = \mathbf{U} \log(\mathbf{\Sigma}) \mathbf{U}^T$, where $\mathbf{R} = \mathbf{U} \mathbf{\Sigma} \mathbf{U}^T$, $\log(\mathbf{\Sigma})$ is the diagonal matrix of the eigenvalue logarithms. However, the spectral operation not only suffers from the problem of zeroes occurring in $\log(\mathbf{\Sigma})$ due to the property of the rotation matrix \mathbf{R} , but also consumes too much time for matrix gradient computation [23]. Therefore, we resort to other approaches to perform the function of this

layer. Fortunately, we can explore the relationship between the logarithm map and the axis-angle representation as:

$$\log(\mathbf{R}) = \begin{cases} 0, & \text{if } \theta(\mathbf{R}) = 0, \\ \frac{\theta(\mathbf{R})}{2\sin(\theta(\mathbf{R}))}(\mathbf{R} - \mathbf{R}^T), & \text{otherwise,} \end{cases} \quad (12)$$

where $\theta(\mathbf{R})$ is the angle of the rotation matrix \mathbf{R} . With this equation, the corresponding matrix gradient can be easily derived by traditional element-wise matrix calculation.

4.4. Output Layers

After performing the LogMap layer, the outputs can be transformed into vector form and concatenated directly frame by frame within one sequence due to their Euclidean nature. To handle the Euclidean forms, we can add regular network layers. For example, the rectified linear unit (ReLU) layer and the regular fully connected (FC) layer could be employed. In particular for the ReLU layer, we can simply set relatively small elements to zero as done in classical ReLU. In the FC layer, the dimensionality of the weight is set to $d_k \times d_{k-1}$, where d_k and d_{k-1} are the class number and the vector dimensionalities, respectively. We employ a common softmax layer (or softmax log-loss) as the final output layer for the problem of skeleton-based action recognition.

5. Training Procedure

In order to train the proposed LieNet, we exploit the Stochastic gradient descent (SGD) algorithm, which is one of the most popular tools for optimizing deep networks. To begin with, let the model of the LieNet be represented as a sequence of function compositions $f = f^{(l)} \circ f^{(l-1)} \circ f^{(l-2)} \dots \circ f^{(2)} \circ f^{(1)}$ with a parameter tuple $\mathbf{W} = (\mathbf{W}_l, \mathbf{W}_{l-1}, \dots, \mathbf{W}_1)$, where $f^{(k)}$ is the function for the k -th layer, \mathbf{W}_k is the weight parameter of the k -th layer, and l is the number of layers. The loss of the k -th layer could be defined by a function like $L^{(k)} = \ell \circ f^{(l)} \circ \dots \circ f^{(k)}$, where ℓ is the loss function for the final output layer.

To optimize the deep model, one classical SGD algorithm needs to compute the gradient of the objective function, which is typically achieved by the backpropagation chain rule. In particular, the gradients of the weight \mathbf{W}_k and the data \mathbf{X}_{k-1} for the k -th layer can be respectively computed by the chain rule:

$$\frac{\partial L^{(k)}(\mathbf{X}_{k-1}, y)}{\partial \mathbf{W}_k} = \frac{\partial L^{(k+1)}(\mathbf{X}_k, y)}{\partial \mathbf{X}_k} \frac{\partial f^{(k)}(\mathbf{X}_{k-1})}{\partial \mathbf{W}_k}, \quad (13)$$

$$\frac{\partial L^{(k)}(\mathbf{X}_{k-1}, y)}{\partial \mathbf{X}_{k-1}} = \frac{\partial L^{(k+1)}(\mathbf{X}_k, y)}{\partial \mathbf{X}_k} \frac{\partial f^{(k)}(\mathbf{X}_{k-1})}{\partial \mathbf{X}_{k-1}}, \quad (14)$$

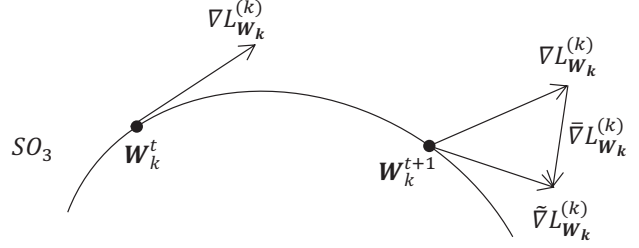


Figure 3. Parallel transport of a tangent vector $\nabla L_{\mathbf{W}_k}^{(k)}$ from a point \mathbf{W}_k^t to another point \mathbf{W}_k^{t+1} on the SO_3 manifold.

where y is the class label, $\mathbf{X}_k = f^{(k)}(\mathbf{X}_{k-1})$. Eqn.13 is the gradient for updating \mathbf{W}_k , while Eqn.14 computes the gradients in the layers below to update \mathbf{X}_{k-1} .

The gradients of the data involved in RotPooling, LogMap and regular output layers can be calculated by Eqn.14 as usual. In particular, the gradient for the data in RotPooling can be computed with the same gradient computing approach used in a regular max pooling layer in the context of traditional ConvNets. For the data in the LogMap layer, the gradient can be obtained by the element-wise gradient computation on the involved rotation matrices.

On the other hand, the computation of the gradients of the parameter weights defined in the RotMap layers is non-trivial. This is because the weight matrices are enforced to be on the Riemannian manifold SO_3 of the rotation matrices, i.e. the Lie group. As a consequence, merely using Eqn.13 to compute their Euclidean gradients rather than Riemannian gradients in the procedure of backprop would not generate valid rotation weights. To handle this problem, we propose a new approach of updating the weights used in Eqn.6 for the RotMap layers. As studied in [1], the steepest descent direction for the used loss function $L^{(k)}(\mathbf{X}_{k-1}, y)$ with respect to \mathbf{W}_k on the manifold SO_3 is the Riemannian gradient $\tilde{\nabla} L_{\mathbf{W}_k}^{(k)}$, which can be obtained by parallel transporting the Euclidean gradients onto the corresponding tangent space. For a better intuition, Fig.3 shows the process of the parallel transport. In particular, transporting the gradient from a point \mathbf{W}_k^t to another point \mathbf{W}_k^{t+1} requires subtracting the normal component $\nabla \bar{L}_{\mathbf{W}_k}^{(k)}$ at \mathbf{W}_k^{t+1} , which can be obtained as follows:

$$\tilde{\nabla} L_{\mathbf{W}_k}^{(k)} = \nabla L_{\mathbf{W}_k}^{(k)} \mathbf{W}_k^T \mathbf{W}_k, \quad (15)$$

where the Euclidean gradient $\nabla L_{\mathbf{W}_k}^{(k)}$ is computed by using Eqn.13 as

$$\nabla L_{\mathbf{W}_k}^{(k)} = \frac{\partial L^{(k+1)}(\mathbf{X}_k, y)}{\partial \mathbf{X}_k} \mathbf{X}_{k-1}^T. \quad (16)$$

Thanks to the parallel transport, the Riemannian gradient can be calculated by

$$\tilde{\nabla} L_{\mathbf{W}_k}^{(k)} = \nabla L_{\mathbf{W}_k}^{(k)} - \nabla \bar{L}_{\mathbf{W}_k}^{(k)}. \quad (17)$$

Searching along the tangential direction takes the update in the tangent space of the SO_3 manifold. Then, such update is mapped back to the SO_3 manifold with a retraction operation. Consequently, an update of the weight \mathbf{W}_k on the SO_3 manifold is of the following form

$$\mathbf{W}_k^{t+1} = \Gamma(\mathbf{W}_k^t - \lambda \tilde{\nabla} L_{\mathbf{W}_k}^{(k)}), \quad (18)$$

where \mathbf{W}_k^t is the current weight, Γ is the retraction operation, λ is the learning rate.

6. Experiments

In order to measure LieNet’s performance for skeleton-based action recognition, we conduct experiments on three standard 3D human action datasets.

6.1. Evaluation Datasets

G3D-Gaming dataset [5] contains 663 sequences of 20 different gaming motions such as tennis serve, golf swing, bowling. Each subject performed every action more than two times. In addition, the 3D locations of 20 joints (i.e., 19 bones) are provided with the dataset.

HDM05 dataset [30] consists of 2,337 sequences of 130 action classes executed by various actors. Most of the motion sequences have been performed several times by all five actors according to the guidelines in a script. As G3D-Gaming [5] dataset, 3D locations of 31 joints (i.e., 30 bones) of the subjects are provided as well with this dataset.

NTU RGB+D dataset [36] is, to the best of our knowledge, currently the largest 3D action recognition dataset. Different from most datasets, it is collected by Kinect v2 and contains more than 56,000 action sequences. A total of 60 different action classes are performed by 40 subjects. The 3D coordinates of 25 joints (i.e., 24 bones) are also offered with this dataset. Due to its large scale, the dataset is highly suitable for deep learning.

6.2. Implementation Details

For the feature extraction, we use the source code of [41] to represent a 3D human skeleton with relative 3D geometry relations between all pairs of body parts. As introduced before, each time instance (frame) of a 3D human skeleton can be represented as a point on the Lie group $SO_3 \times \dots \times SO_3$. As preprocessed in [41], we normalize any sequence of motion into a fixed N -length one, with an equal number of samples. As a consequence, we finally get 100, 16, 64 frames for G3D-Gaming, HDM05 and NTU RGB-D datasets, respectively. In this setting, each motion sequence can be represented as a curve of length N on the Lie group $SO_3 \times \dots \times SO_3$.

As the focus of this work is on skeleton-based action recognition, we mainly utilize manifold-based approaches for comparison. The two baseline approaches are the special Euclidean group (SE) [40] and the special orthogonal group (SO) [41] representations based on shallow learning methods. For a fair comparison, we use the source codes from the original authors, and set the involved parameters as in the original papers. For the proposed LieNet, we build its architecture with single or multiple block(s) of RotMap/RotPooling layers illustrated in Fig.1 before the three final layers, that are LogMap, FC and softmax layers. The learning rate λ is fixed to 0.01, the batch size is set to 30, the weights in the RotMap layers are initialized as random rotation matrices, the number of samples for the skeleton-based level RotPooling layer is set to 4. For training the LieNet, we just use an i7-6700K (4.00GHz) PC without any GPUs. As the LieNet gets promising results on all datasets with the same configuration, this shows its insensitivity to the parameter settings. Note that, for the LieNet, we do not employ the dynamic time warping (DTW) technique [29], which has been used in the SO and SE methods to solve the problem of speed variations.

6.3. Experimental Results

G3D-Gaming dataset [5]. For the dataset, we follow a cross-subject test setting, where half the subjects are used for training and the other half are employed for testing. All the results reported for this dataset are averaged over ten different combinations of training and testing datasets.

Table 1 compares the proposed LieNet with the state-of-the-art methods (i.e., RBM-HMM [31], SE [40] and SO [41]) reported for the G3D-Gaming dataset. For fair comparison, we report their results without using the Fourier Temporal Pyramid (FTP) post-processing (their accuracies are 91.09% and 90.94% after using FTP). As shown in Table 1, the LieNet shows its superiority over the two baseline methods SO and SE. Besides, as we can see, our LieNet with 3 blocks of RotMap and RotPooling layers achieves the best performance. For this dataset, we also study the performances of different block numbers in the LieNet architecture. As the number of frames in each sequence was fixed to 100 as mentioned before, using 3 blocks has pooled each sequence into 7 frames. Thus, we study 3 blocks at most for our LieNet. As observed from Table 1, the improvement by adding more blocks demonstrates the effectiveness of stacking more RotMap/RotPooling blocks.

In addition, we evaluate the performances of different LieNet configurations as shown in Fig.4. The left of Fig.4 verifies the necessity of using RotMap, RotPooling and LogMap layers to improve the proposed LieNet-3Blocks. In addition, we also compare the LieNet with and with-

Method	G3D-Gaming
RBM+HMM [31]	86.40%
SE [40]	87.23%
SO [41]	87.95%
LieNet-0Block	84.55%
LieNet-1Block	85.16%
LieNet-2Blocks	86.67%
LieNet-3Blocks	89.10%

Table 1. Experimental results (recognition accuracies) on the G3D-Gaming database.

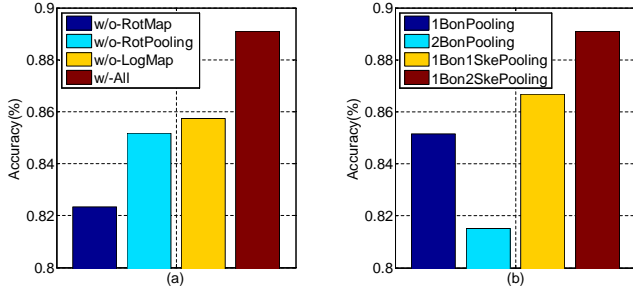


Figure 4. (a) Comparison of different LieNet configurations: without using RotMap layers (w/o-RotMap), w/o-RotPooling layers, w/o-LogMap layers and using all (w/-All) in LieNet-3Blocks for G3D-Gaming. (b) Comparison of different pooling schemes: using 1 bone-based pooling layer (1BonPooling, i.e., LieNet-1Block), 2 bone-based pooling layers (2BonPooling), 1BonPooling+1 skeleton-based pooling layer (1Bon1SkePooling, i.e., LieNet-2Blocks), 1BonPooling+2SkePooling (1Bon2SkePooling, i.e., LieNet-3Blocks) for G3D-Gaming.

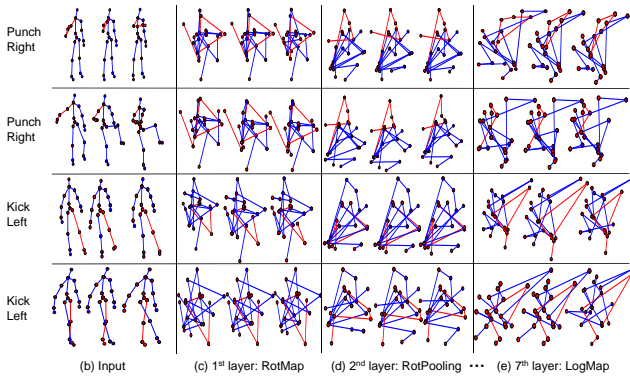


Figure 5. The example skeletons are reconstructed by the output rotation matrices of some representative LieNet layers for processing four action sequences from the G3D-Gaming dataset. The bones in red are the interesting ones for the action classes.

out DTW. On this dataset, the performance (88.89% vs. 89.10%) of these two cases is approximately equal. Therefore, the benefit of using RotMap layers somehow shows it can take the role of DTW that solves the problem of speed

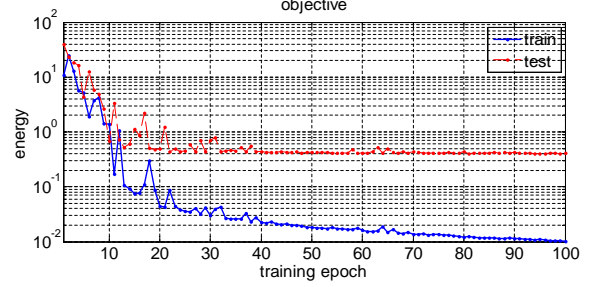


Figure 6. The convergence behavior of the proposed LieNet for the G3D-Gaming dataset.

variations. The right of Fig.4 analyses the effectiveness of the 1BonPooling case (i.e., Fig.2(a)→(b)), and shows the decreasing performance behavior of the 2BonPooling case (i.e., Fig.2(a)→(b)→(c)). Thus, we finally utilize 1BonPooling and 2SkePooling (i.e., Fig.2(c)→(d)) in the LieNet-3Blocks structure. Besides, we also study the behavior of adding a rectified linear unit (ReLU)-like layer (i.e., setting the matrix elements below a threshold $\epsilon = 0.1$ to zero) on the top of the LogMap layer as presented before. Yet, the performance was worse (87.58%) than without. Further, to validate the improvements are from the contribution of the RotMap and RotPooling layers rather than deeper architectures, we build a regular (LeNet-like) deep structure, i.e., LogMap→2×(FC→MaxPooling)→FC→ReLU→FC→Softmax, that applies 8 regular layers on the concatenated output Euclidean forms of the LogMap layer. The step for MaxPooling is set to 4, and the sizes of different FC weights are set to 307800×40000 , 10000×4000 , 1000×400 and 400×20 respectively. The performance of this network is 85.49%, which supports the validation.

For a better understanding of the proposed LieNet, we also visualize the output results of some representative layers. In particular, we roughly estimate the 3D location of each body bone, given the learned rotation matrix and the 3D coordinate of the beginning edge in the torso part. In Fig.5, we present the visualization of some layers for four action sequences, that belong to the classes of ‘punch right’ and ‘kick left’. As shown in Fig.5, we observe that they yield meaningful semantic information layer by layer for specific classes. Specifically, the reconstructions from the first layer (RotMap) and the second layer (RotPooling) typically still mix some patterns specific for the action classes with some rather confusing ones. But, when arriving at the the seventh layer (LogMap), the patterns for specific motion classes become more discriminative.

Finally, we also empirically analyze the convergence behavior of the exploited training procedure, even though the convergence of the used SGD algorithm on Riemannian manifolds has been studied well in [8, 6] already. As de-

Method	HDM05
SPDNet [18]	61.45% \pm 1.12
SE [40]	70.26% \pm 2.89
SO [41]	71.31% \pm 3.21
LieNet-0Block	71.26% \pm 2.12
LieNet-1Block	73.35% \pm 1.14
LieNet-2Blocks	75.78%\pm2.26

Table 2. Experimental results (recognition accuracies) on the HDM05 database.

Method	RGB+D-subject	RGB+D-view
HBRNN [13]	59.07%	63.97%
Deep RNN [36]	56.29%	64.09%
Deep LSTM [36]	60.69%	67.29%
PA-LSTM [36]	62.93%	70.27%
ST-LSTM [25]	69.2%	77.7%
SE [40]	50.08%	52.76%
SO [41]	52.13%	53.42%
LieNet-0Block	53.54%	54.78%
LieNet-1Block	56.35%	60.14%
LieNet-2Blocks	58.02%	62.52%
LieNet-3Blocks	61.37%	66.95%

Table 3. Experimental results (recognition accuracies) for cross-subject and cross-view evaluations on the NTU RGB+D database.

picted in Fig.6, the training algorithm can converge to a stable solution after 100 epochs.

HDM05 dataset [30]. Following [18], we conduct 10 random evaluations, each of which randomly selects half of the sequences for training and the rest for testing.

As listed in Table 2, besides to the two baseline methods SE and SO, we also study the SPDNet method that has reached the best performance so far for this dataset. The large improvement of SE and SO over SPDNet suggests the effectiveness of the Lie group representations for the problem of skeleton-based action recognition. As the last experiment on the G3D-Gaming dataset, we also study the proposed LieNet with different numbers of blocks of RotMap and RotPooling layers. Note that as the length of each sequence in this database is fixed to 16 frames, we implemented the LieNet with 2 blocks at most. As reported in Table 2, using more blocks improves over using less blocks, and gets the state-of-the-art on the dataset, again showing its advantages over SE and SO shallow learning methods.

NTU RGB+D dataset [36]. This dataset has two standard testing protocols. One is cross-subject test, for which half of the subjects are used for training and the rest is for testing. The other one is cross-view test, for which two views are employed for training and the rest one is utilized for testing.

Since this dataset is large enough to train deep networks, recent works [36, 25] studied typical Recurrent Neural Networks (deep RNN and deep LSTM) as well as two variants, i.e., part-aware (PA) and spatio-temporal (ST) versions of LSTM. The common advantage of these deep networks is to learn temporal information, and to significantly outperform the Lie group representation learning methods SE and SO, which are good at learning spatial information, but are no deep learning models. In this paper, our LieNet fills the gap by showing the effectiveness of deep learning for the spatial representations. As shown in Table 3, our LieNet with more stacked blocks can significantly improve the two baseline methods SE and SO, which validates the effectiveness of the deep learning. By comparing with the state-of-the-art methods on this database, our LieNet behaves better or equally well as most deep networks (e.g., deep RNN and deep LSTM) that exploit temporal information. The LieNet is still outperformed by the recently proposed networks (PA-LSTM and ST-LSTM) however, which jointly learn spatial and temporal features for skeletal motion sequences. This is reasonable because the LieNet is mainly designed to learn the spatial features.

7. Summary and Future Work

We studied a deep network architecture in the domain of Lie group features, that is successful for skeleton-based action recognition. In order to handle the key issues of speed variation and high dimensionality of the Lie group features, we designed special mapping layers and pooling layers to process the resulting rotation matrices. In addition, we also exploited logarithm mapping layers to perform Riemannian computing on the representations, with which regular output layers are supplied in the new network structure. The final evaluations on three standard 3D action datasets not only demonstrated the effectiveness of the proposed network, but also compared its different configurations. Moreover, we also showed an interesting visualization for the network, which somewhat discloses its intrinsic mechanism.

As the proposed network is, to the best of our knowledge, the first attempt to perform deep learning on Lie groups for skeleton-based action recognition, there are quite a few open issues. For example, studying multiple rotation mappings per RotMap layer and exploiting a ReLU-like layer in the context of a Lie group network are worth paying attention to. Besides, building a deeper network, beginning from the raw 3D joint locations up to the Lie group features in an end-to-end learning manner, could be more effective. Last but not least, encouraged by the success of the deep spatio-temporal networks [36, 25], exploring the potential of the proposed network in the temporal setting would also be an interesting direction.

References

- [1] P. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2008.
- [2] F. Albertini and E. D. Sontag. For neural networks, function determines form. In *Decision and Control, 1992., Proceedings of the 31st IEEE Conference on*, pages 26–31. IEEE, 1992.
- [3] R. Anirudh, P. Turaga, J. Su, and A. Srivastava. Elastic functional coding of Riemannian trajectories. *IEEE T-PAMI*, 2016.
- [4] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.
- [5] V. Bloom, D. Makris, and V. Argyriou. G3D: A gaming action dataset and real time action recognition evaluation framework. In *CVPR Workshops*, 2012.
- [6] S. Bonnabel. Stochastic gradient descent on Riemannian manifolds. *IEEE Trans. Auto. Control*, 58(9):2217–2229, 2013.
- [7] D. Boscaini, J. Masci, S. Melzi, M. Bronstein, U. Castellani, and P. Vandergheynst. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. In *Computer Graphics Forum*, volume 34, pages 13–23. Wiley Online Library, 2015.
- [8] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT*, 2010.
- [9] N. Boumal and P.-A. Absil. A discrete regression method on manifolds and its application to data on $SO(n)$. *IFAC Proceedings Volumes*, 44(1):2284–2289, 2011.
- [10] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014.
- [11] R. Chaudhry, A. Ravichandran, G. Hager, and R. Vidal. Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In *CVPR*, 2009.
- [12] D. L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.
- [13] Y. Du, W. Wang, and L. Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *CVPR*, 2015.
- [14] S. Fiori. Unsupervised neural learning on Lie group. *International Journal of Neural Systems*, 12(03n04):219–246, 2002.
- [15] R. Gens and P. M. Domingos. Deep symmetry networks. In *NIPS*, 2014.
- [16] B. C. Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*. Springer, 2015.
- [17] F. Han, B. Reily, W. Hoff, and H. Zhang. Space-time representation of people based on 3D skeletal data: a review. *arXiv preprint arXiv:1601.01006*, 2016.
- [18] Z. Huang and L. Van Gool. A Riemannian network for SPD matrix learning. *arXiv preprint arXiv:1608.04233*, 2016.
- [19] Z. Huang, R. Wang, S. Shan, and X. Chen. Projection metric learning on Grassmann manifold with application to video based face recognition. In *CVPR*, 2015.
- [20] Z. Huang, R. Wang, S. Shan, X. Li, and X. Chen. Log-Euclidean metric learning on symmetric positive definite manifold with application to image set classification. In *ICML*, 2015.
- [21] K. Hüper and F. S. Leite. On the geometry of rolling and interpolation curves on $S(n)$, $SO(n)$, and Grassmann manifolds. *Journal of Dynamical and Control Systems*, 13(4):467–502, 2007.
- [22] M. E. Hussein, M. Torki, M. A. Gawayyed, and M. El-Saban. Human action recognition using a temporal hierarchy of covariance descriptors on 3D joint locations. In *IJCAI*, 2013.
- [23] C. Ionescu, O. Vantzos, and C. Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *ICCV*, 2015.
- [24] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena. Structural-RNN: Deep learning on spatio-temporal graphs. In *CVPR*, 2016.
- [25] J. Liu, A. Shahroudy, D. Xu, and G. Wang. Spatio-temporal LSTM with trust gates for 3D human action recognition. In *ECCV*, 2016.
- [26] F. Lv and R. Nevatia. Recognition and segmentation of 3D human action using HMM and multiclass adaboost. In *ECCV*, 2006.
- [27] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *ICCV Workshops*, 2015.
- [28] Y. Moreau and J. Vandewalle. A Lie algebraic approach to dynamical system prediction. In *Circuits and Systems, 1996. ISCAS'96., Connecting the World., 1996 IEEE International Symposium on*, volume 3, pages 182–185. IEEE, 1996.
- [29] M. Müller. *Information retrieval for music and motion*, volume 2. Springer, 2007.
- [30] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation: Mocap database HDM05. *Tech. Rep. CG-2007-2*, 2007.

- [31] S. Nie and Q. Ji. Capturing global and local dynamics for human action recognition. In *ICPR*, 2014.
- [32] D. W. Pearson. Changing network weights by Lie groups. In *Artificial Neural Nets and Genetic Algorithms*, pages 249–252. Springer, 1995.
- [33] D. W. Pearson. Hopfield networks and symmetry groups. *Neurocomputing*, 8(3):305–314, 1995.
- [34] L. L. Presti and M. La Cascia. 3D skeleton-based human action classification: A survey. *Pattern Recognition*, 53:130–147, 2016.
- [35] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
- [36] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang. NTU RGB+ D: A large scale dataset for 3D human activity analysis. In *CVPR*, 2016.
- [37] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.
- [38] P. Turaga and R. Chellappa. Locally time-invariant models of human activities using trajectories on the Grassmannian. In *CVPR*, 2009.
- [39] A. Veeraraghavan, A. K. Roy-Chowdhury, and R. Chellappa. Matching shape sequences in video with applications in human movement analysis. *IEEE T-PAMI*, 27(12):1896–1909, 2005.
- [40] R. Vemulapalli, F. Arrate, and R. Chellappa. Human action recognition by representing 3D skeletons as points in a Lie group. In *CVPR*, 2014.
- [41] R. Vemulapalli and R. Chellappa. Rolling rotations for recognizing human actions from 3D skeletal data. In *CVPR*, 2016.
- [42] C. Wang, Y. Wang, and A. L. Yuille. An approach to pose-based action recognition. In *CVPR*, 2013.
- [43] C. Wang, Y. Wang, and A. L. Yuille. Mining 3D key-pose-motifs for action recognition. In *CVPR*, 2016.
- [44] J. Wang, Z. Liu, Y. Wu, and J. Yuan. Mining actionlet ensemble for action recognition with depth cameras. In *CVPR*, 2012.
- [45] R. Zbikowski. Lie algebra of recurrent neural networks and identifiability. In *American Control Conference*, number 30, pages 2900–2901, 1993.